# Parallel Visibility Computations for Parallel Radiosity

by

W. Stürzlinger and C. Wild

Johannes Kepler University of Linz
Institute for Computer Science
Department for graphical and parallel Processing
Altenbergerstraße 69, A-4040 Linz, Austria, Europe

Tel.: +43(732)2468-884, Fax : +43(732)2468-10
e-mail: wrzl@gup.uni-linz.ac.at

# Parallel Visibility Computations for Parallel Radiosity

W. Stürzlinger and C. Wild

Institute for Computer Science, Johannes Kepler University of Linz, Austria

**Abstract**

The radiosity method models the interaction of light between diffuse reflecting surfaces, thereby accurately predicting global illumination effects. Due to the high computational effort to calculate the transfer of light between surfaces and the memory requirements for the scene description, a distributed, parallelized verison of the algorithm is needed for scenes consisting of thousands of surfaces. We present a distributed, parallel radiosity algorithm, which can subdivide the surfaces adaptively. Additionally we present a scheme for parallel visibility calculations. Adaptive load redistribution is also discussed.

## 1 Introduction

Radiosity has become a popular method for image synthesis due to its ability to generate images of high realism. It was first introduced to computer graphics by Goral et al. [GORA84]. Further research resulted in the progressive refinement method, which is able to produce good approximations of the final solution very quickly [COHE88]. The radiosity method was extended to include specular reflection through the so called two-pass approach. For recent developements see [MALL88, SILL89, SILL91].

Common to all these methods is the representation of the surfaces of the environment by a mesh of quadrilaterals and triangles. These "patches" are used to store the radiosity on the respective part of the surface.

The geometric formfactors were first calculated by the use of a hemicube [COHE85]. A hemicube is placed around the center of a patch and all other patches are projected onto its surfaces. The projected area gives an estimate for the geometric formfactor between the patches.

Because this estimate of the formfactors may be inexact even for simple cases [BAUM89], other methods for computing the formfactors were suggested and/or implemented. Baum used a hybrid method involving both numerical and analytic methods [BAUM89] and other methods use raytracing to compute the formfactors [WALL89, SILL89, MALL88, TAMP91]. Wallace subdivides the shooting patch until an analytic solution to approximate the formfactor of the delta-areas can be used. Then the formfactors for all visible delta-areas are summed up giving a good approximation to the formfactor of the shooting patch.

The calculation of the formfactors accounts for most of the computation time of the radiosity method. Also the memory requirement is clearly a function of the number of patches. These problems led to the developement of parallel implementations of the progressive refinement radiosity method [BAUM90, RECK90, FEDA91, CHAL91].

### 1.1 Progressive Refinement

The radiosity method partitions the surfaces of the scene in small, flat patches and computes the illumination for each of those patches. The radiosity of a patch is determined by the radiosity it emits directly plus all light that is reflected. This is described by the radiosity equation:

$$B_i = E_i + \rho_i \sum_{j=0}^{n-1} F_{i,j} B_j \tag{1}$$

where

- n is the number of the patches.
- $B_i$ is the radiosity of the $i$-th patch.
- $E_i$ is the emitted radiosity of the $i$-th patch.
- $\rho_i$ is the reflectivity of the $i$-th patch.
- $F_{i,j}$ is the formfactor from patch $i$ to patch $j$.

The linear equation system defined above can be solved with e.g. an iterative Gauss-Seidel solution method and this method converges quickly in practice. As the memory usage to store the formfactor matrix is proportional to $n^2$ this method becomes impractical for larger $n$. The progressive refinement method solves this equation system iteratively also. Due to a reordering of the solution process, it is only neccesary to calculate (and store) one column of the matrix per iteration step. The iteration then distributes ("shoots") the radiosity of the patch with the maximum unshot radioisity to all other patches in the environment.

## 2 Parallelization of the Progressive Refinement Method

### 2.1 Previous Research

Parallelization of the Progressive Refinement Method has been atempted in several ways. Baum [BAUM90] used a multiprocessor workstation calculating the hemicubes using a hardware z-buffer. Recker [RECK90] used a cluster of workstations.

Feda [FEDA91] presented an implementation on a transputer network, where each processor has local memory. The formfactors were calculated using the hemicube method. Chalmers [CHAL91] also presented an implementation on a transputer network. He improved the accuracy of the formfactor calculation by using the analytical approch described by [BAUM89].

The use of the hemicube or the analytical method still suffers from the problem that the formfactors and the visibilities are determined using the "shooting"-patch as projection center. This leads to noticeable artifacts. A better method is to calculate this information directly for each receiver.

In the following sections we assume that we have a number of processors with local memory and that we have a interconnecting network. We did our tests on an nCube with 64 nodes with 4 MB memory each, connected by a hypercube topology network, but we emphasize that the algorithm can be ported easiliy to other architectures.

### 2.2 Parallel Progressive Refinement

This paper presents an approach based on the calculation of formfactors by raytracing as described by Wallace [WALL89]. Raytracing is used to determine the visible parts of the "shooting"-patch as seen from each patch. The formfactor of these visible parts is then calculated using the analytical solution to the contour integtral. The visibility is determined by subdividing the "shooting"-patch regularily into $M$ parts and tracing a ray from the receiver to each of these parts.

In the following discussion we assume that each processor stores the data for all patches. Therefore the number of patches is limited by the available memory on each processor. In section 3 we will show how this algorithm can be extended for larger numbers of patches. Then the following steps are performed until the solution has converged:

- All processors send their "best" patch to the master processor, i.e. the patch which has the most unshot radiosity.
- The master processor chooses the globally best patch as "shooter" and sends this information to all processors. Note that the "shooter"-geometry and its radiosity data are distributed to all processors in this step.
- The following steps are performed on all processors in parallel:
  - For each receiving patch we determine the visibility of the "shooting"-patch by tracing rays to the shooter. Each ray is intersected with all other patches. For the visibile parts we calculate the formfactor and add the resulting contribution to the receiver radiosity.

In contrast to many previously presented algorithms, we don't have to update radiosity values on other processors, we just communicate to determine the "shooter"-patch.

### 2.3 Rendering

After the solution has converged we render the scene. This can be done e.g. by sending all patches to the workstation and using its hardware z-buffer to render the picture. An alternative is to render the picture in parallel also. This could be done as follows:

We partition the picture into $N$ parts and assign each part to one of the $N$ processors, we can use a ring topology to distribute all patches to all processors, or exploit the network topology more fully. The load distribution will be quite uneven for this method, as different parts of the picture will "contain" significantly different numbers of patches. This has been reported by Feda [FEDA91] but they also propose a scheme which assigns every $N$-th scanline to a processor thus distributing the load much more evenly.

## 3 Parallel Visibility Calculation

In section 2.2 we presented a parallel progressive radiosity algorithm. One shortcoming of the method of section 2.2 is, that the number of patches is limited by the available memory on each processor. This is due to the fact, that we have to intersect each visibility ray with all other patches to determine the visibility of the "shooter"-patch.

Each processor which calculates and stores the radiosity of patches is called radiosity processor from now on, and processors assigned to visibility calculations are called visibility processors.

Let us assume that we assign a number of visibility processors to each radiosity processor. The patches of the scene are distributed evenly among each group of visibility processors. I.e. the union of the patches stored on the visibility processors assigned to one radiosity processor is equal to the set of patches of the scene.

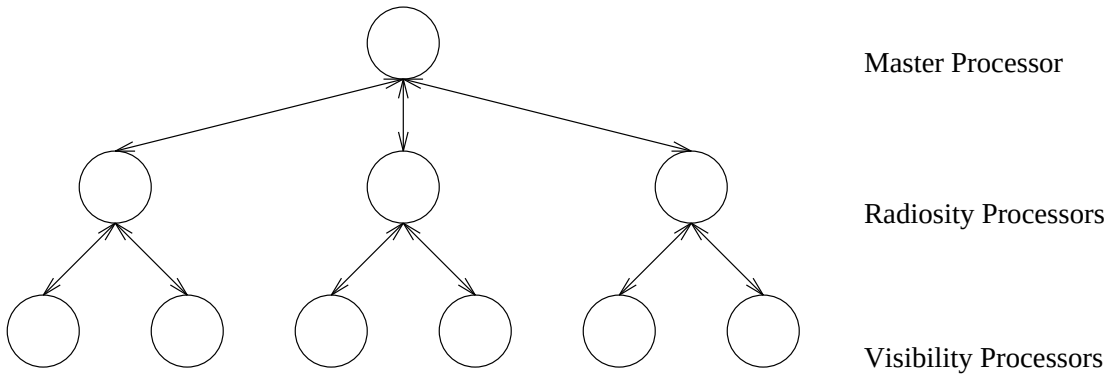


Figure 1: Communication structure for 2 visibility processors assigned to each of 3 radiosity processors.

In other words we store the scene four times in the above configuration, three times split into each of the visibility processor pairs and one time split into the three radiosity processors.

The following algorithm also assumes that we use a fixed subdivision of the "shooter"-patch, e.g. into 16 or 64 parts.

- We send a point, for which the visibility is to be determined, from the radiosity processor to its visibility processors.
- Each visibility processor determines for each part of the subdivision of the "shooter"-patch, if it is visible or not and returns the result in a bit vector to its radiosity processor.
- The radiosity processor collects all response messages from its visibility processors and binary-and's the bit vectors together. The resulting bit vector describes now which parts of the "shooter"-patch are visible. All bits, which are set, correlate to a part of the "shooter"-patch which was determined visible on all visibility processors. All zero bits correlate to parts invisible on at least one visibility processor, i.e. at least one patch obstructed the part of the "shooter"-patch. The ratio of set bits to the number of parts of the subdivision describes now the visibility of the "shooter"-patch.

To reduce the communication overhead, we transfer the points, for which the visibilities are to be determined, in blocks of e.g. 100 to the visibility processors. Also the resulting bit vectors are transfered in identical sized blocks back to the radiosity processor.

# 4 Patches and Elements

The two-level hierarchy of patches proposed by [COHE86] can also be used with this method.

We subdivide each patch into elements and store these elements locally on the processor. The visibility and formfactor calculations now need to be done for each element, therefore we will have to communicate more frequently with the visibility processors. The visibility processors still store only the geometry for "their" patches for the intersection tests.

Additional communication overhead will also appear during the rendering phase, as all elements have to be transfered (and rendered).

For a better load distribution we distribute the patches so that an approximate equal number of elements has to be handled by each radiosity processor.

### 4.1 Adaptive Subdivision of the Environment

This approach can be generalized by subdividing the elements adaptively, e.g. if the radiosity values at the corners of an element are too different. The test is done after the energy has been distributed to all elements. The adaptive subdivision of elements progresses recursively until all elements are (almost) uniformly lit.

In the parallel version adaptive subdivision is a local decision, therefore there is no additional communication overhead, except for a greater number of visibility tests.

### 4.2 Dynamic Load Balancing

Of course this scheme is not optimal from the standpoint of load distribution. Some processors with deeply subdivided surfaces (i.e. a great number of elements) will have a significantly higher load for all further interactions than others. As we find that a processor has a significantly higher number of elements than other processors, we can transfer some patches and the associated elements to a less loaded processor.

This strategy can also be used if we run out of memory while apaptively subdiving.

## 5 Implementation and Results

Our approach was implemented on an nCube2 with 64 processors, which is a distributed memory computer where the nodes are connected with a hypercube topology network. The performance of a single processor is 2.5 MFLOPS.

As a basis we used the progressive radiosity algorithm for patches and elements. We parallelized this algorithm as desribed in section 2.2 and implemented the parallel visibility calculation method (section 3). For practical reasons we used the first radiosity processor as master processor.

All given times are in seconds for one iteration of the progressive radiosity algorithm. In the tables R denotes the number of radiosity processors, V the number of visibility processors and T denotes the total number of processors.

We used the following scenes for out tests.

| Name | Patches | Elements |
|---|---|---|
| Scene 0 | 52 | 1162 |
| Scene 1 | 288 | 4648 |
| Scene 2 | 832 | 18952 |
| Scene 3 | 3328 | 74368 |

The first test was performed with a varying number of radiosity processors without visibility processors.

| R | V | T | Scene 1 | Scene 2 |
|---|---|---|---|---|
| 4 | 0 | 4 | 170 | 2728 |
| 8 | 0 | 8 | 107 | 1361 |
| 16 | 0 | 16 | 60 | 683 |
| 32 | 0 | 32 | 14 | 427 |
| 64 | 0 | 64 | 4 | 244 |

The time decreases sublinearily due to the fact that all processors are synchronized implicitely by the best "shooter"-patch selection.

Then a varying number of radiosity processors with three visibility processors each was tested.

| R | V | T | Scene 1 | Scene 2 | Scene 3 |
|---|---|---|---|---|---|
| 4 | 3 | 16 | 64 | 993 | - |
| 8 | 3 | 32 | 39 | 483 | 8436 |
| 16 | 3 | 64 | 23 | 244 | 3882 |

Again the time decreases sublinearily, but this time the synchronization process is not as noticeable as in the previous test.

The next test was performed with a varying number of visibility processors per radiosity processor.

| R | V | T | Scene 1 | Scene 2 |
|---|---|---|---|---|
| 4 | 0 | 4 | 170 | 2728 |
| 4 | 1 | 8 | 175 | 2770 |
| 4 | 3 | 16 | 64 | 993 |
| 4 | 7 | 32 | 31 | 463 |
| 4 | 15 | 64 | 19 | 252 |

The performance increases with the number of visibility processors as expected, but the increase is not linear as the communication overhead increases with the number of visibility processors.

Another test was performed with a fixed number of total processors and different configurations.

| R | V | T | Scene 1 | Scene 2 | Scene 3 |
|---|---|---|---|---|---|
| 4 | 15 | 64 | 19 | 252 | (3811) |
| 8 | 7 | 64 | 18 | 222 | 3781 |
| 16 | 3 | 64 | 23 | 244 | 3882 |
| 32 | 1 | 64 | 14 | 433 | 5513 |
| 64 | 0 | 64 | 4 | 244 | 2727 |

We could not obtain data for scene 3 with 4 radiosity and 15 visbility processors because the radiosity processors ran out of memory for this configuration (not enough memory for elements). Therefore the time is given for a configuration with 5 radiosity and 11 visibility processors for a total of 60.

The configurations with no visibility processors perform quite well. Too small a number of visibility processors slows the algorithm down, as the radiosity processor has to wait for the visibility processors to finish.

For 7 visibility processors (i.e. 8 radiosity processors) we have a local minimum. For scene 1 this is also the optimum.

Greater numbers of visibility processors also decrease the performance due to the greater communication overhead.

The communication overhead for the visibility calculations is quite noticeable, especially for small scenes, i.e. scene 1. Here it is a lot quicker to perform the visibility test locally than to wait for the data from the visibility processor to return.

For scene 0 we also measured the total time, i.e. the time the progressive radiosity algorithm took to reach convergence. The progressive radiosity algorithm performs 138 iteration until convergence on all configurations.

| R | V | T | Scene 0 |
|---|---|---|---|
| 1 | 0 | 1 | 4861 |
| 4 | 7 | 32 | 426 |
| 8 | 3 | 32 | 503 |
| 32 | 0 | 32 | 457 |
| 4 | 15 | 64 | 299 |
| 8 | 7 | 64 | 278 |
| 16 | 3 | 64 | 306 |
| 52 | 0 | 52 | 371 |

As scene 0 consists of 52 patches we were not able to exploit the 64 processor nodes fully. Interestingly enough the fastest time was reported for the configuration with 8 radiosity and 7 visibility processors.

# 6 Conclusion and Further Extensions

The newly presented method has a number of advantages.

- Formfactor calculation is done by raytracing, which proved to deliver much more accurate results than the hemicube method.
- There is no need to distribute radiosity values after formfactor calculation, as in previous parallelization attempts. Therefore all calculations can be done locally, and no communication is necessary in this phase.
- The method allows adaptive subdivision of the surfaces. Additionally the uneven load which results from the adaptive subdivision can be redistributed using the scheme described in section 4.2.
- The parallel visibility calculation method allows us to render scenes where the memory to store the patches and elements exceeds the available memory per processor node. We are currently modelling some scenes, where even the memory for the patches exceeds available memory. i.e. $\geq$ 15000 patches. As the results indicate, for certain processor configurations the method also outperforms the naive parallelization of the progressive radiosity method.

In the current version the visibility test is quite inefficient, as we don't use any optimization method to speed up the ray tracing. We are currently implementing an octree method

to improve the performance of the visibility tests. This should speed up the whole algorithm significantly. Especially we expect, that configurations with 3 to 7 visibility processors perform substantially better.

All processors are synchronized at every progressive radiosity iteration by the best "shooter"-patch selection, therefore the performance is determined by the slowest processor. Detecting such a processor and transfering some of its patches (and the assoicated elements) to other processors should improve the performance of following progressive radiosity iterations considerably.

## References


[BAUM89] Daniel R. Baum, Holly E. Rushmeier, James M. Winget, *"Improving Radiosity Solutions through the Use of Analytically Determined Form-Factors"*, Computer Graphics (SIGGRAPH '89 Proceedings), July 1989.

[BAUM90] Daniel R. Baum, James M. Winget, *"Real Time Radiosity Through Parllel Processing and Hardware Acceleration"*, Computer Graphics (SIGGRAPH '90), July 1990.

[CHAL91] Alan G. Chalmers, Derek J. Paddon, *"Parallel Processing of Progressive Refinement Radiosity Methods"*, in Proceedings of the Second Eurographics Workshop on Rendering, May 1991.

[COHE85] Michael Cohen, Donald P. Greenberg, *"The Hemi-Cube: A Radiosity Solution for Complex Environments"*, Computer Graphics (SIGGRAPH '85 Proceedings), August 1985.

[COHE86] Michael Cohen, Donald P. Greenberg, Dave S. Immel, Phillip J. Brock, *"An Efficient Radiosity Approach for Realistic Image Synthesis"*, IEEE Computer Graphics and Applications, March 1986.

[COHE88] Michael Cohen, Shenchang Eric Chen, John R. Wallace, Donald P. Greenberg, *"A Progressive Refinement Approach to Fast Radiosity Image Generation"*, Computer Graphics (SIGGRAPH '88 Proceedings), August 1988.

[FEDA91] Martin Feda, Werner Purgathofer, *"Progressive Refinement Radiosity on a Transputer Network"*, in Proceedings of the Second Eurographics Workshop on Rendering, May 1991.

[GORA84] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, Bennett Battaile, *"Modelling the Interaction of Light Between Diffuse Surfaces"*, Computer Graphics (SIGGRAPH '84 Proceedings), July 1984.

[MALL88] Thomas J.V. Malley, *"A Shading Method for Computer Generated Images"*, Master's Thesis, University of Utah, June 1988.

[RECK90] Rodney J. Recker, David W. George, Donald P. Greenberg, *"Acceleration technique for Progressive Refinement Radiosity"*, Computer Graphics (SIGGRAPH '90), July 1990.

[SILL89] Francois Sillion, Claude Puech, *"A General Two-Pass Method Integrating Specular and Diffuse Reflection"*, Computer Graphics (SIGGRAPH '89 Proceedings), July 1989.

[SILL91] Francois X. Sillion, James R. Arvo, Stephen H. Westin, Donald P. Greenberg, *"A Global Illumination Solution for General Reflectance Distributions"*, Computer Graphics (SIGGRAPH '91 Proceedings), July 1991.

[TAMP91] F. Tampieri, D. Lischinski, *"The Constant Radiosity Assumption Syndrome"*, in Proceedings of the Second Eurographics Workshop on Rendering, May 1991.

[WALL89] John R. Wallace, Kells A. Elmquist, Eric A. Haines, *"A Ray Tracing Algorithm for Progressive Radiosity"*, Computer Graphics (SIGGRAPH '89 Proceedings), July 1989.